The Importance of Accounting for Execution Failures when Predicting Test Flakiness

Presented by:
Guillaume HABEN, University of Luxembourg, Luxembourg

Authors:

Guillaume Haben	University of Luxembourg, Luxembourg
Sarra Habchi,	Ubisoft, Canada
John Micco,	Broadcom, USA
Mark Harman,	University College London & META, UK
Mike Papadakis,	University of Luxembourg, Luxembourg
Maxime Cordy,	University of Luxembourg, Luxembourg
Yves Le Traon	University of Luxembourg, Luxembourg

Date:
October 30th, 2024

PayPal

Serval
securityandtrust.lu

SNT
securityandtrust.lu

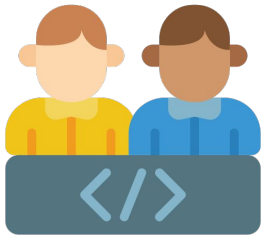
uni.lu
UNIVERSITY OF LUXEMBOURG

Jim

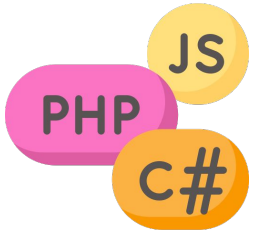

Google

Google

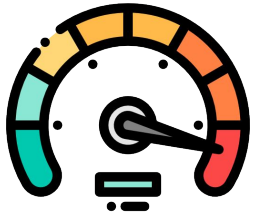
>10k software engineers



>100m lines of code projects



>1 000 commits per hour



Challenges

Testing Large Software Systems

Handle Large Amounts of Tests

Hundreds of thousands of tests

Avoid Anti-Patterns

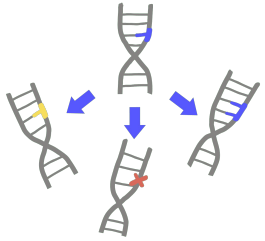
Regression Test Selection

Test Case Prioritization

Ensure Test Quality

Test Coverage

Test Robustness

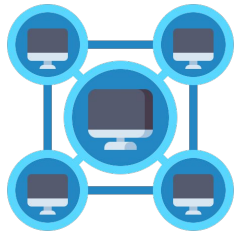


Test Refactoring

Manage Multi-Environments

Dev, Test, Prod

Distributed Testing



Platform Dependencies

Definition

“A flaky test is a test that

can both pass or fail when executed several times

on the same version of a program”

Why does it matter?

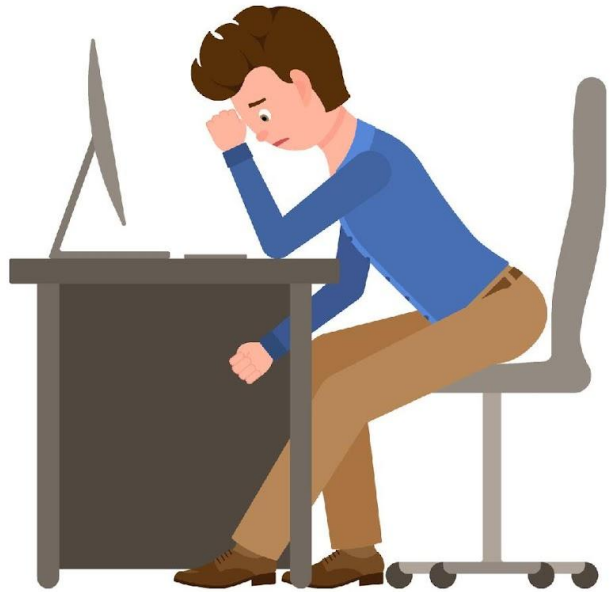
Flaky tests often accounts for 1-5%

Flakiness increases costs both time-wise and computer-wise

At Google: up to 16% of testing budget spent just to rerun flaky tests

This leads to technical debt, bad quality and impacts other testing strategies relying on deterministic tests

Major problem in modern software testing



C. Leong, A. Singh, M. Papadakis, Y. Le Traon, J. Micco, Assessing transition-based test selection algorithms at google, ICSE 2019

J. Micco, The State of Continuous Integration Testing @Google, ICST 2017

Concrete example of a flaky test

```
# https://github.com/python-telegram-bot/python-telegram-bot/blob/master/tests/test_updater.py
def test_idle(self, updater, caplog):
        updater.start_polling(0.01)
        Thread(target=partial(self.signal_sender, updater=updater)).start()
        with caplog.at_level(logging.INFO):
            updater.idle()
        rec = caplog.records[-2]
        assert rec.getMessage().startswith('Received signal {signal.SIGTERM}')
        assert rec.levelname == 'INFO'
        rec = caplog.records[-1
        assert rec.getMessage().startswith('Scheduler has been shut down')
        assert rec.levelname == 'INFO'
        # If we get this far, idle() ran through
        sleep(0.5)
        assert updater.running is False
```

Motivation

Current Research on Flakiness Prediction

Study	Model	Feature category	Features	Benchmark	Target	Year
King et al. [91]	Bayesian network	Static & dynamic	Code metrics	Industrial	**Flaky tests**	2018
Pinto et al. [92]	Random forest	Static	**Vocabulary**	DeFlaker	**Flaky tests**	2020
Bertolino et al. [93]	KNN	Static	**Vocabulary**	DeFlaker	**Flaky tests**	2020
Haben et al. [94]	Random forest	Static	**Vocabulary**	DeFlaker	**Flaky tests**	2021
Camara et al. [95]	Random forest	Static	**Vocabulary**	iDFlakies	**Flaky tests**	2021
Alshammari et al. [96]	Random forest	Static & dynamic	Code metrics & Smells	FlakeFlagger	**Flaky tests**	2021
Fatima et al. [97]	Neural Network	Static	CodeBERT	FlakeFlagger iDFlakies	**Flaky tests**	2021
Pontillo et al. [98]	Logistic regression	Static	Code metrics & Smells	iDFlakies	**Flaky tests**	2021
Lampel et al. [99]	XGBoost	Static & dynamic	Job execution metrics	Industrial	Flaky failures	2021
Qin et al. [100]	Neural Network	Static	Dependency graph	Industrial	**Flaky tests**	2022
Olewicki et al. [101]	XGBoost	Static	**Vocabulary**	Industrial	Flaky builds	2022
Ackli et al. [102]	Siamese Networks	Static	CodeBERT	Various	**Flaky tests**	2022

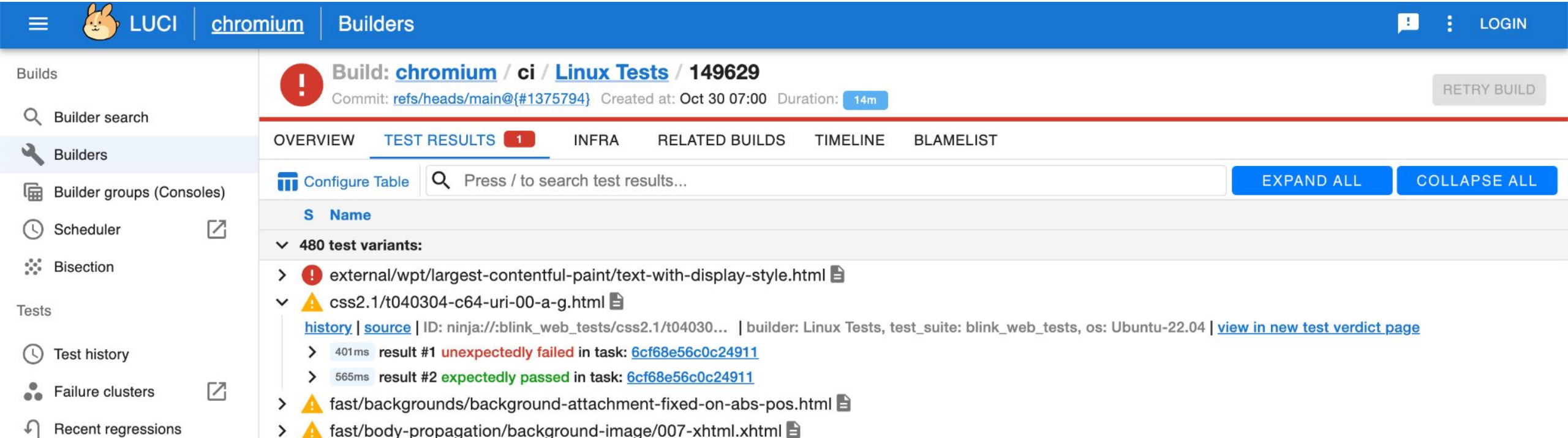
Most of the previous research focuses on predicting flaky tests using vocabulary features

Case study: Chromium

Large project with its own custom CI Framework: LuCI

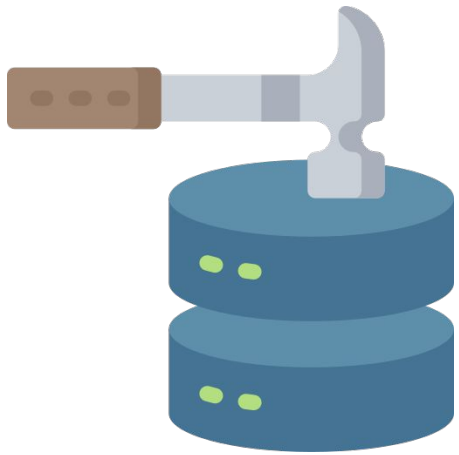
~80 million LoC

Built for hundreds of OS and versions



Definitions

Builds



1 build: specific revision

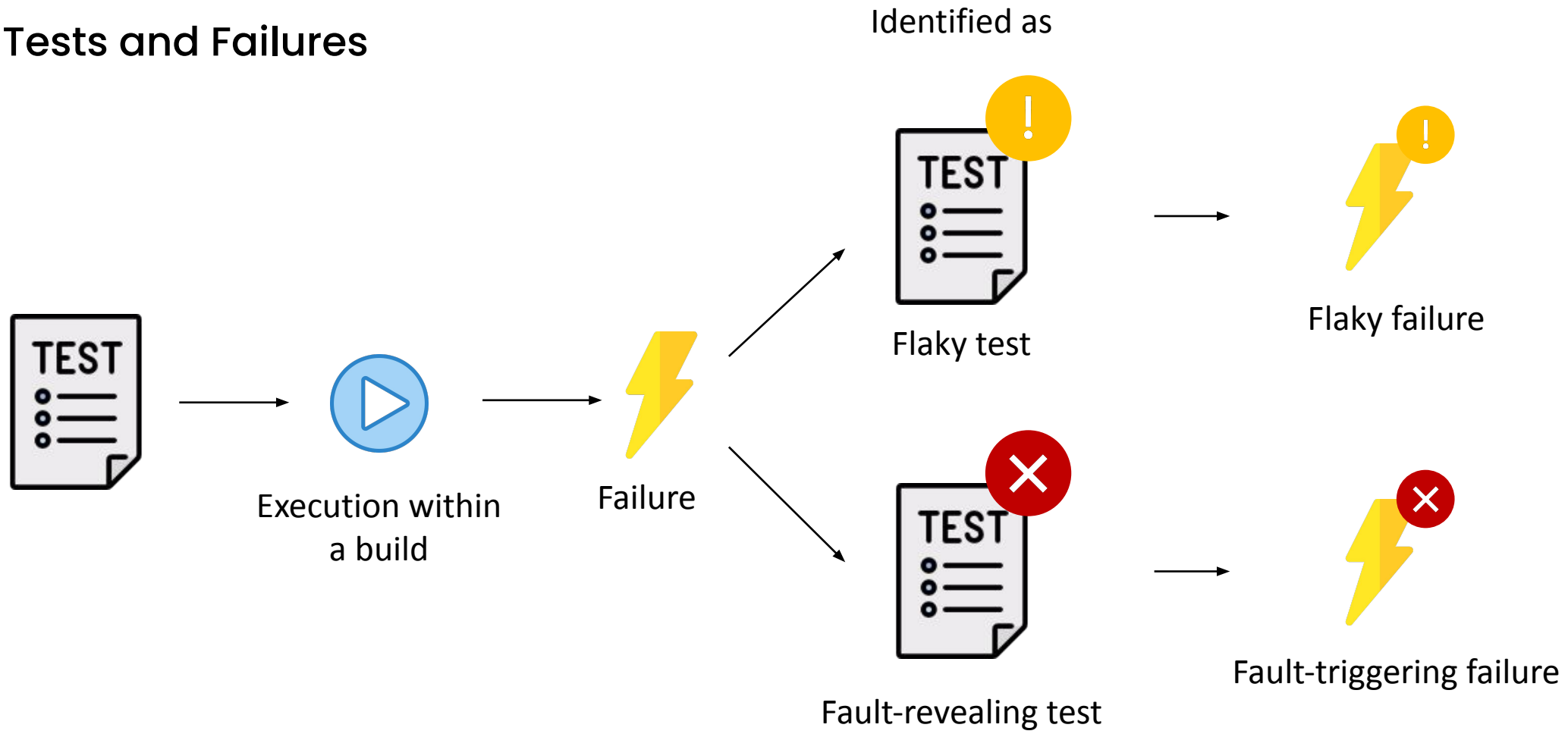
Either:

Builder: Compiles the project
- Specific version, instrumentations, OS

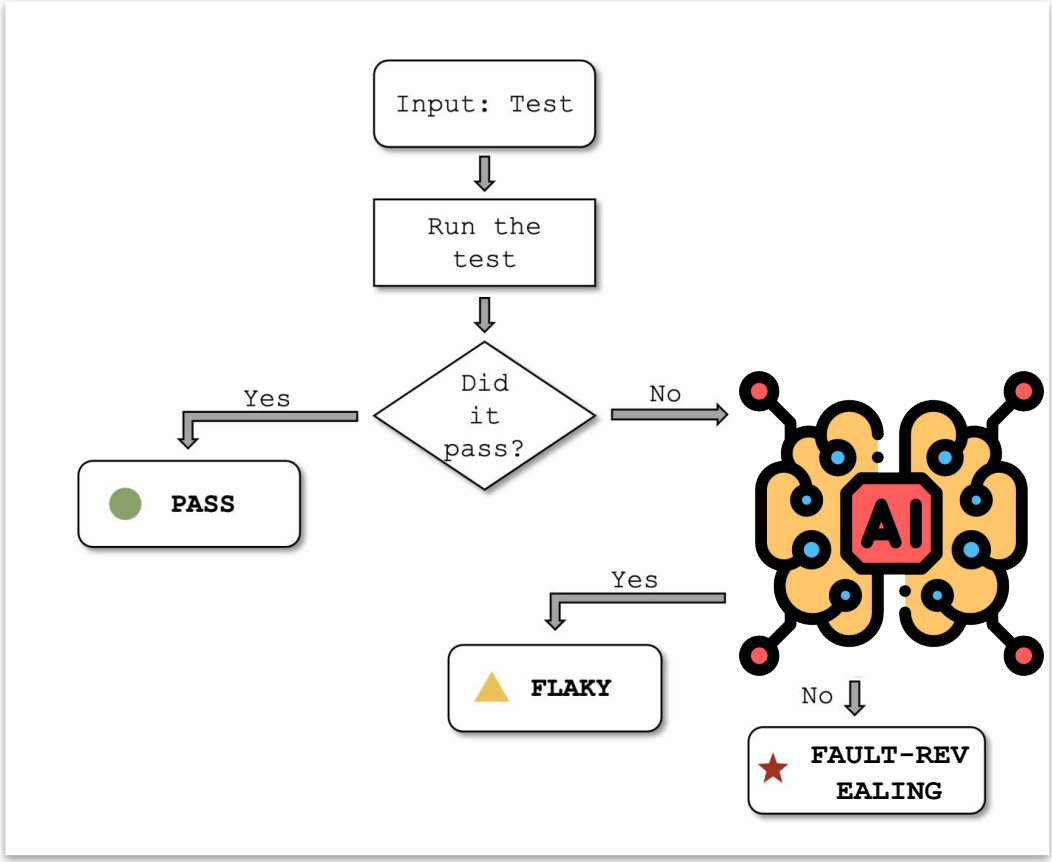
Tester: Runs regression tests
~200,000 tests (unit, integration, GUI)

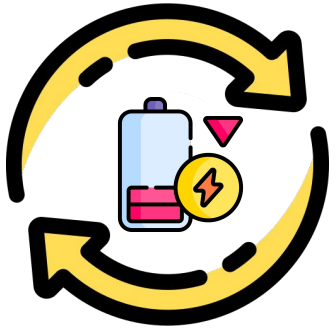
Definitions

Tests and Failures



Identifying flaky tests

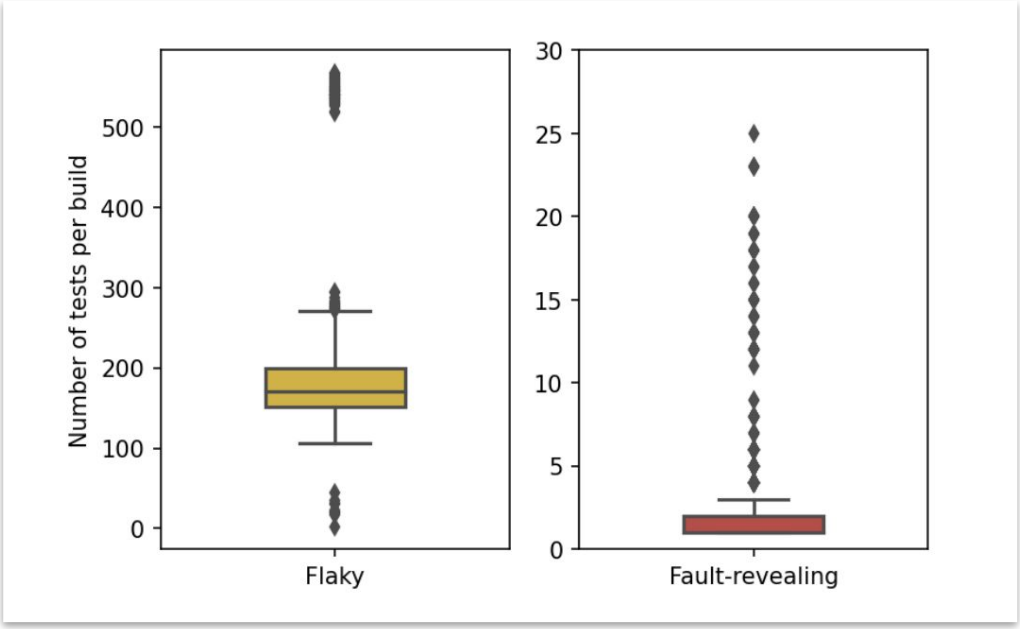




Data collection

Dataset

Tester	Nb of Builds	Period of Time		Number of Tests			Number of Failures	
		From	To	Passing	Flaky	Fault-revealing	Flaky	Fault-triggering
Linux Tests	10,000	Mar 2, 2022	Dec 1, 2022	198,273	23,374	2,343	1,833,831	17,171



Number of flaky and fault-revealing test per builds

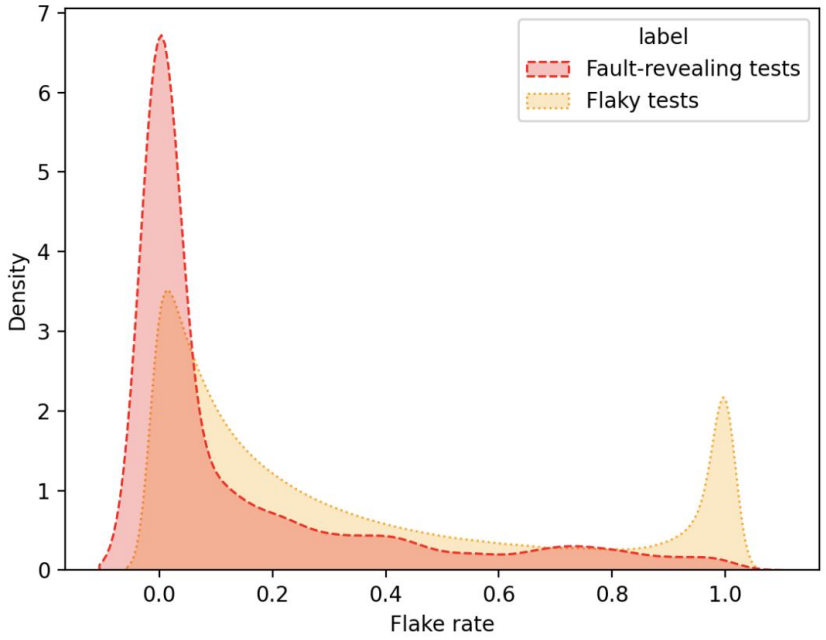
Question

Are current approaches appropriate to classify test failures?

Retrieved features

Feature Name	Feature Description
buildId	The build number associated with the test execution
flakeRate	The flake rate of the test over the last 35 builds
runDuration	The time spent for this test execution
runStatus	ABORT FAIL PASS CRASH SKIP
runTagStatus	CRASH PASS FAIL TIMEOUT SUCCESS FAILURE FAILURE_ON_EXIT NOTRUN SKIP UNKNOWN
testSource	The test source code
testSuite	The test suite the test belongs to
testId	The test name

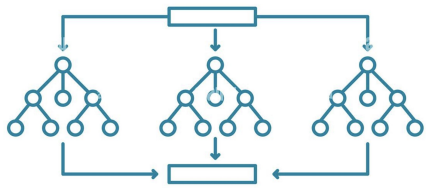
The flake rate is often used in the industry to quantify the level of flakiness of a test



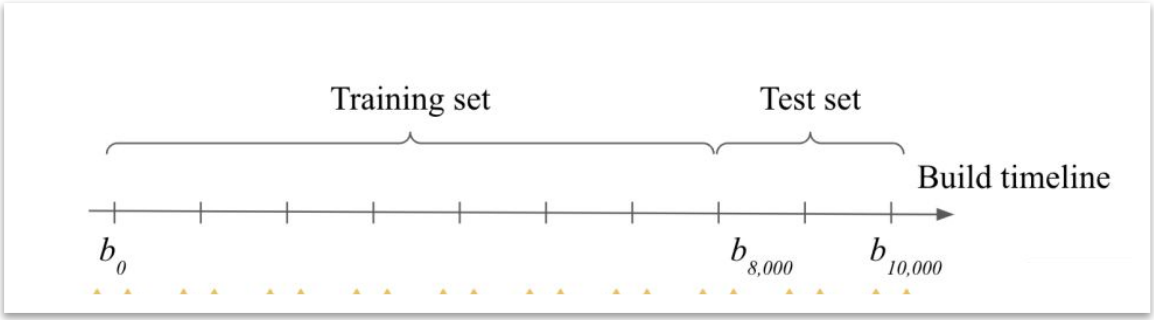
Experimental settings

Model training

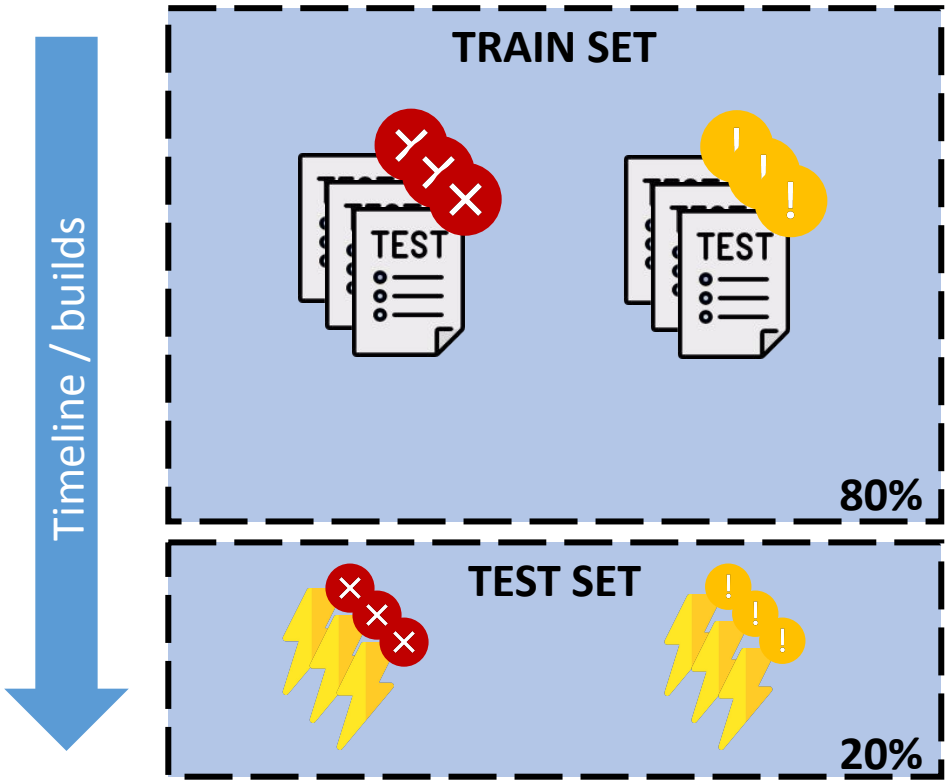
Random Forest Classifier



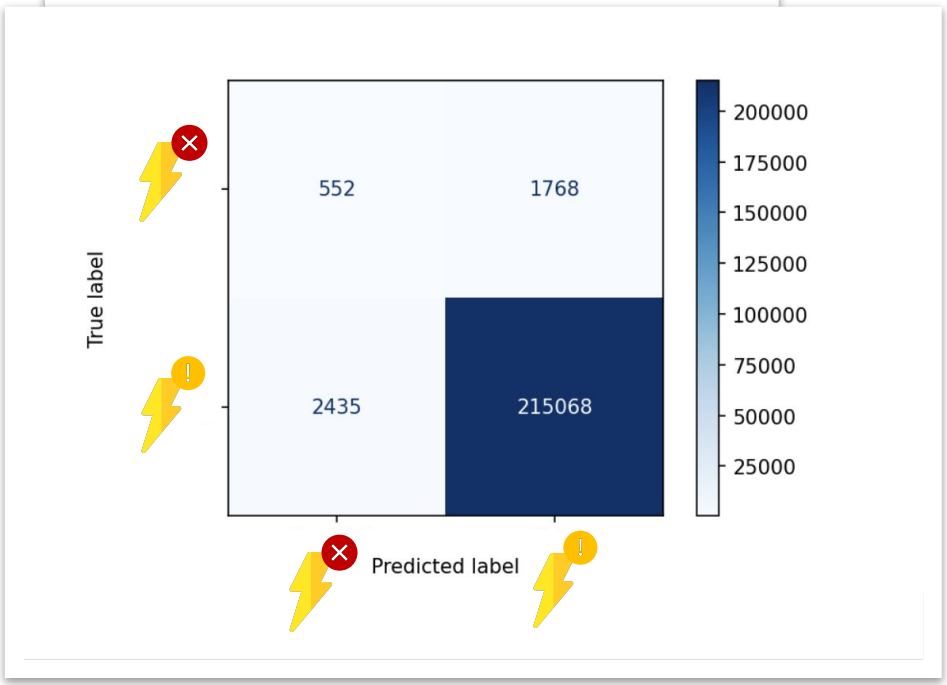
Time-sensitive analysis



Performance of existing approaches

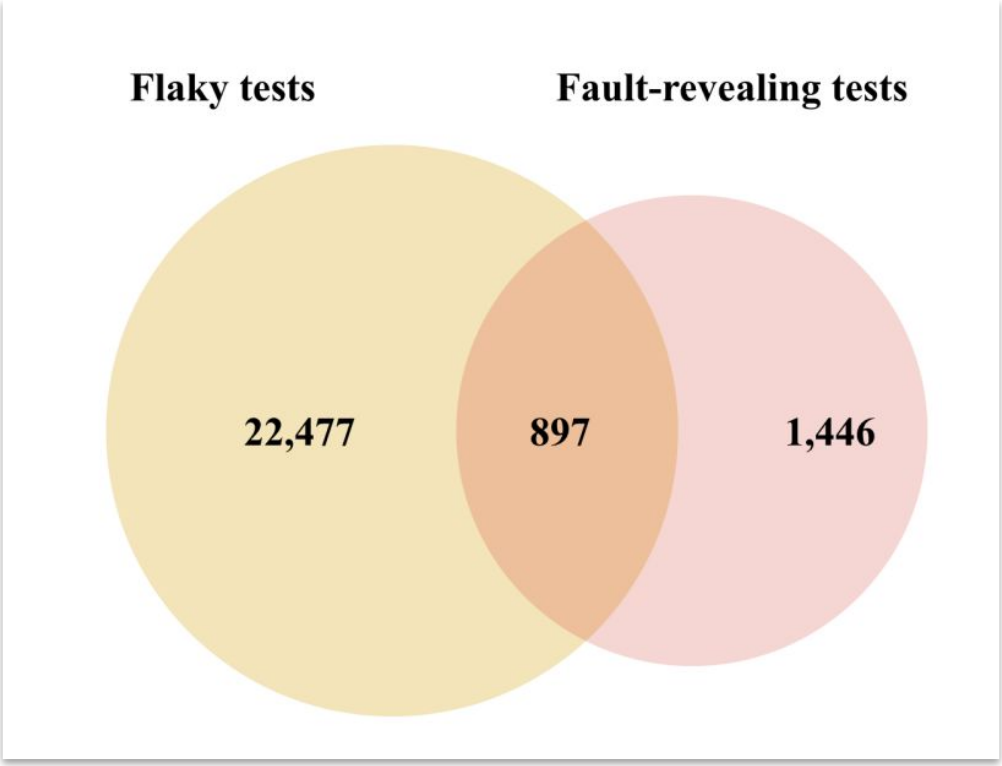


Precision	Recall	MCC	FPR
99.2%	98.9%	0.20	76.2%



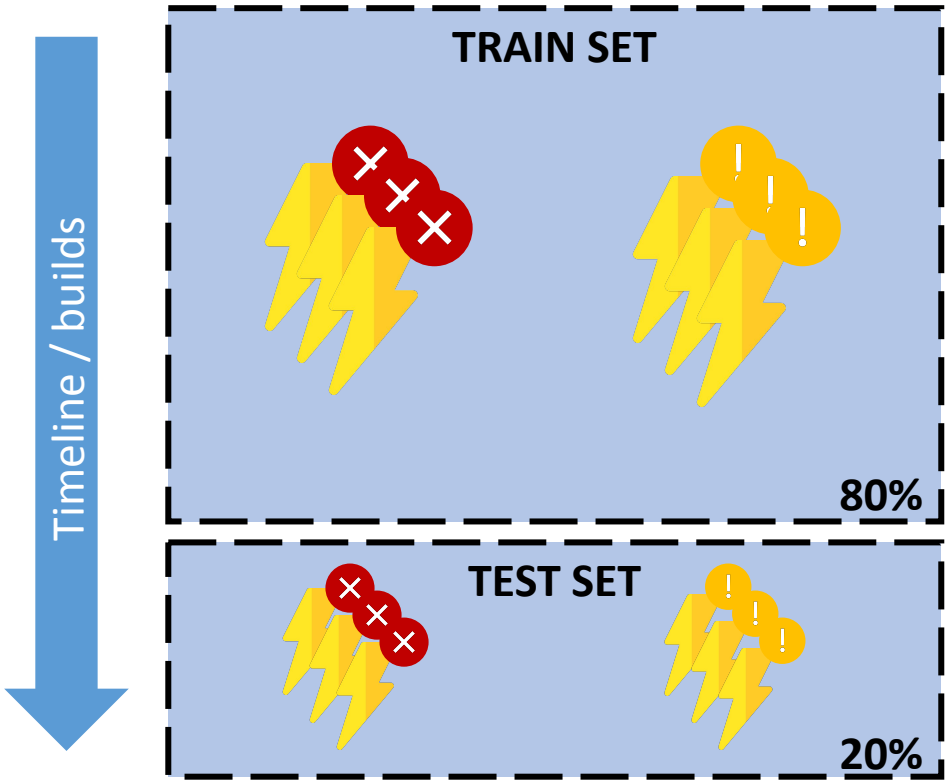
¾ of fault-triggering failures are classified as flaky (missed faults)

Cross-build analysis



⅓ of fault-revealing tests were found to be flaky in previous builds

Performance when focusing on failures



Precision	Recall	MCC	FPR
99.7%	91.3%	0.25	20.3%

Training on failures improves the performance but further work is required

Take-away messages

Flaky tests <u>are valuable</u> as they can <u>reveal faults</u>

¾ of fault-triggering failures are misclassified as flaky (missed faults)

Need for execution-focused prediction methods

The Importance of Accounting for Execution Failures when Predicting Test Flakiness

Presented by:
Guillaume HABEN, University of Luxembourg, Luxembourg

Link to paper:

Date:
October 30th, 2024

PayPal

Serval
securityandtrust.lu

SnT
securityandtrust.lu

uni.lu
UNIVERSITY OF LUXEMBOURG